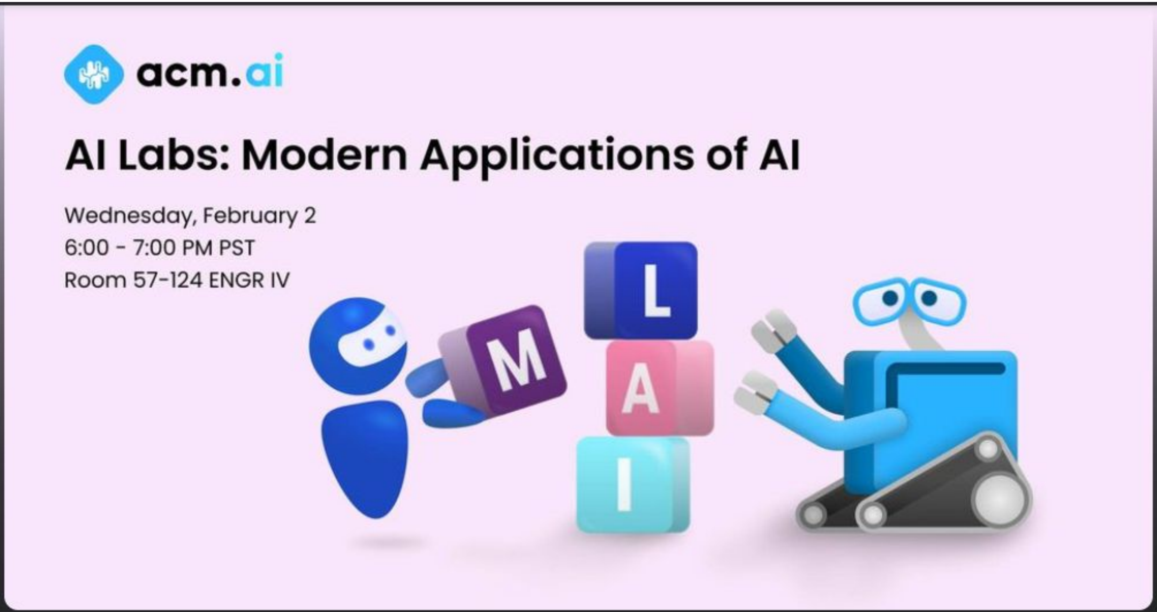
Linear Regression

Intro to Machine Learning: Beginner Track #3

Feedback form: bit.ly/btrack-w22-feedback
Discord: bit.ly/ACMdiscord

acm.ai



2

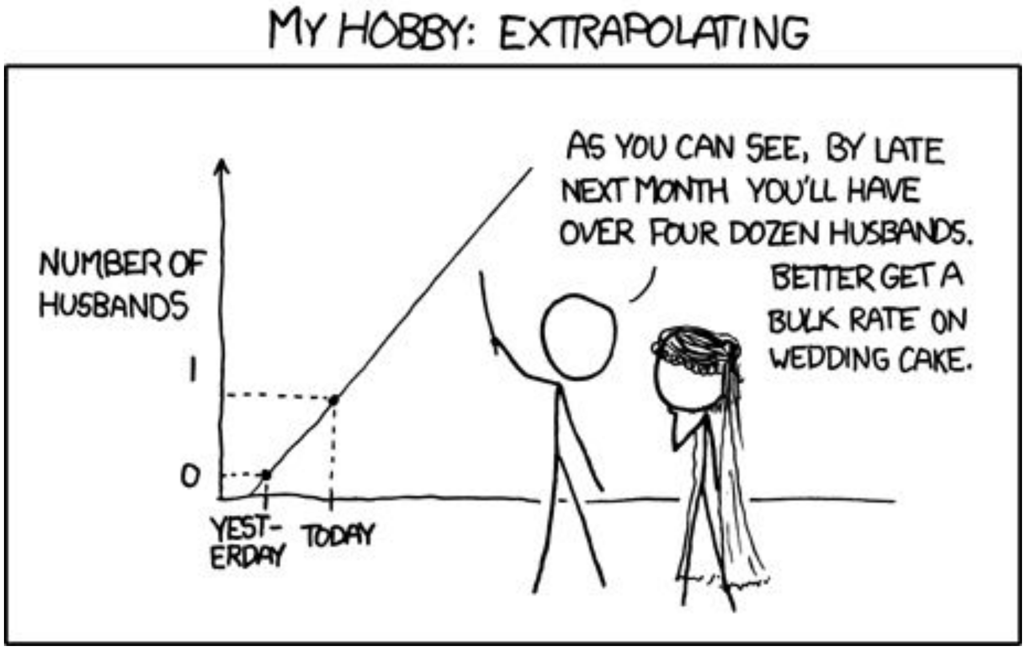
WEDNESDAY, FEBRUARY 2, 2022 AT 6 PM – 7 PM

AI Labs: Modern Applications of AI

Engineering IV Maxwell Room 57-124

Today's Content

- Motivating Example
- Hypothesis Functions
- Some Useful Math
- Loss Function



A Motivating Example

How well will Joe Bruin do on his midterm?

acm acm.ai

Problem: Predicting Your Midterm Performance

- What information might be useful?
  - Time spent studying, Lecture hours attended, etc.
- We call this information about the student: **features**
- The **midterm score** of the student **depends** on these **features.**
- The midterm score becomes the **target**
- What is the relationship between the features and the target?
  - Are they positively/negatively correlated?

How do we represent our data?

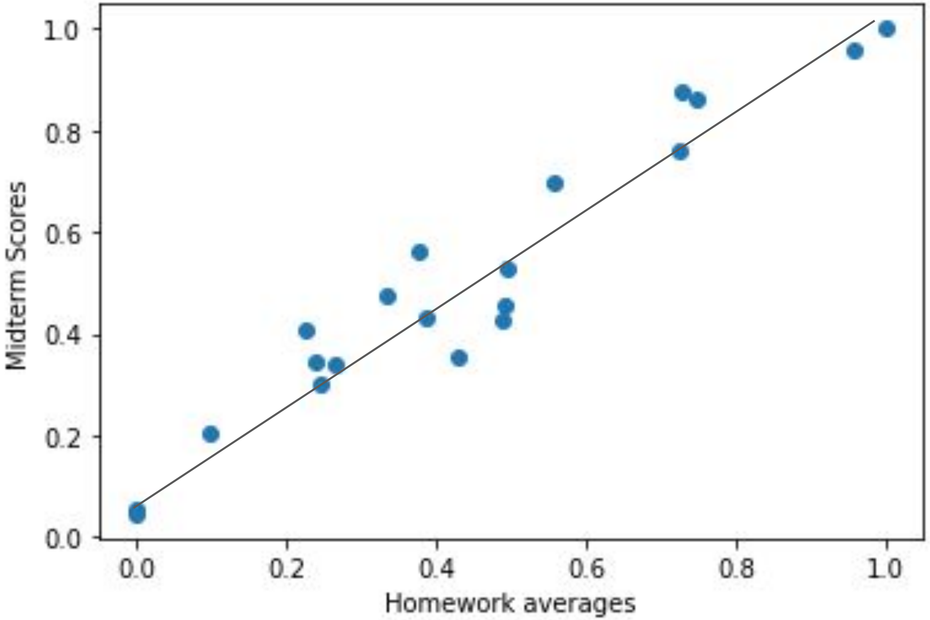
- Each **row** represents the information for **one student** in our data set
- Each **column** represents one **feature**
- The **target** is the list of **midterm scores**. This is what we want to predict. We call this column **y**

y : target

HW Avg	Study Hrs	Lec. Hrs	Midterm %
70%	10	18	85
95%	15	19	90
64%	5	10	60
77%	13	16	76

How would you model this relationship?

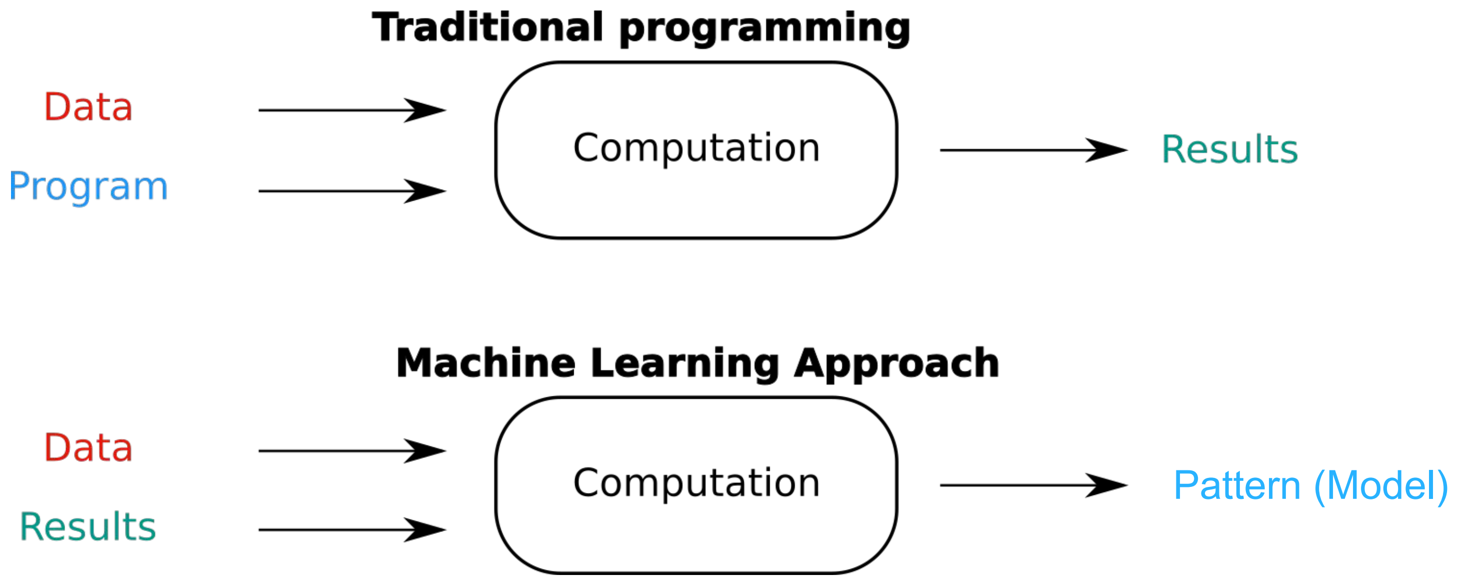
- One way is to use a linear model
- What are the parameters of a line?



The ML way - Pattern Recognition

- We want to predict the midterm score of a student given some features
- ML helps us find a **pattern** that relates the target and the features
- In this case, the pattern is determined by the parameters of our linear model (a line):
    - Slope
    - Bias (Y - intercept)

The ML way vs The Old Fashioned way



Formalizing what we've learned

Hypothesis

- Hypothesis ($\hat{y}$): The function that will **output the score**, given some **features as input**.
- This is what we want to *learn*
- Input features ($x_1, x_2, \ldots, x_n$): The values that plug into ŷ
- Our goal now is to determine **y-hat**
- Mathematically, we want a function: $\hat{y}(x_1, x_2 \ldots x_n)$

acm.ai

How do we use our linear model?

- So we have our input features ($\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$)
- If we want to use a linear model for $\hat{\mathbf{y}}$

$$\hat{y}(x) = b + w_1x_1 + w_2x_2 + \ldots\ldots w_nx_n$$

- Each of the **weights** $w_1, \ldots, w_n$ determine "how much" a particular feature affects the output
- Note: the weights can be negative as well: a high negative weight for the number of classes may negatively impact the score

Weights

$$\hat{y}(x) = b + w_1 x_1 + w_2 x_2 + \ldots\ldots w_n x_n$$

- The **weights** are **w_1, w_2 …. w_n** and **b.**
- They are the **learnable parameters** of our model.
- In the hypothesis above, we can change the function by changing the values of b and **w_1, w_2, …. w_n**
- We need to find the *"best"* possible weights for our model.

Model

$$\hat{y}(x) = b + w_1 x_1 + w_2 x_2 + \ldots\ldots w_n x_n$$

So our model, **ŷ(x)** can also be represented in the following manner :

$$\begin{bmatrix} x_1 & x_2 & x_3 & \ldots & x_n \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_n \end{bmatrix} + b \qquad X \cdot W + b \qquad XW + b$$

Takes the dot product of **X** and **W** vectors, then adds **b**

The parameters

$$\hat{y}(x_1, x_2 \ldots x_n) = b + w_1 x_1 + w_2 x \ldots + w_n x_n$$

An input **X** is an n-dimensional **vector** for the n features in the example

$$[x_1 \; x_2 \; x_3 \; \ldots \; x_n]$$

The weight **W** is also an n-dimensional **vector**

$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_n \end{bmatrix}$$

The bias **b** is a real number

$$b$$

acm acm.ai

Weights - A polling activity

Which feature would impact your midterm score the most?

a. Hours spent studying for midterm
b. Difficulty of the exam
c. Number of classes taken
d. Hours of sleep the night before
e. Number of pets you have

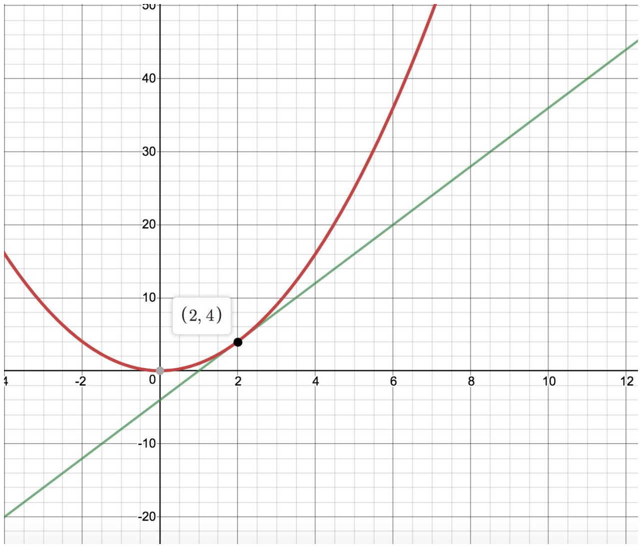
Sample Hypothesis

y-hat(x1,... x5) =

Let’s talk math

Derivatives

- The derivative of a function is the **rate of change** of the function
- If the derivative is **positive**, the function is **increasing**
- If the derivative is **negative,** the function is **decreasing**



Partial Derivatives

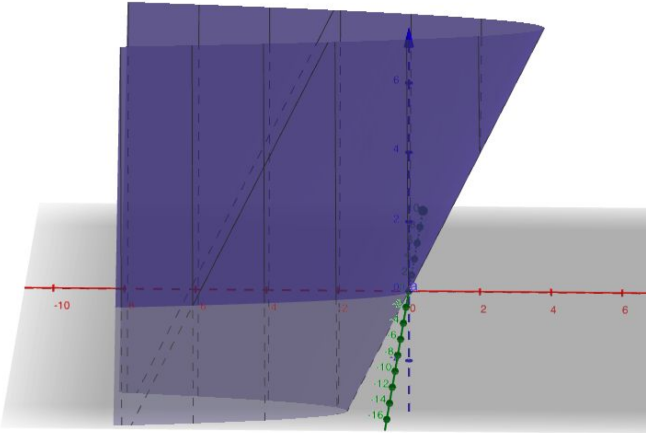
- To take the partial derivative of **f(x,y)** with respect to **x**, we assume **y** to be **constant** and take the derivative as you would for a single variable function

- To take the partial derivative of **f(x_1,x_2,...,x_n)** with respect to some **x_i** we **take every other variable to be constant**, and continue.

$$f(x,y) = 2x + 3y^2$$

$$\frac{\partial f}{\partial x} = 2$$

$$\frac{\partial f}{\partial y} = 6y$$



Calculating a gradient

A gradient of an **n-dimensional function** is an **n-dimensional vector** of the partial derivatives of the function with respect to each variable

$$\nabla f = < \frac{df}{dx}, \frac{df}{dy}, \frac{df}{dz} >$$

$$f(x, y, z) = xsin(y) + 2z^3$$

$$\nabla f(x, y, z) = < sin(y), xcos(y), 6z^2 >$$

Review Questions

What is the gradient of $f(x, y, z) = x^2 + y^2 + z^2$?

a. $\nabla f = \; < 2x,\ 2y,\ 2z >$

b. $\nabla f = \; 2x + 2y + 2z$

c. $\nabla f = \; 1/3 < x^3,\ y^3,\ z^3 >$

d. $\nabla f = \; < x^2,\ y^2, z^2 >$

Loss Function

- In order to “improve” our model, we need to know how accurate we are at any given time
- The **loss function** measures the **error** in your predictions compared to the target values.
- An example of a loss function is **Mean Squared Error (MSE)**

$$L(\hat{y_1}, \hat{y_2}, \ldots \hat{y_m}) = \frac{1}{m} \sum_{i=1}^{m} (y_i - \hat{y_i})^2$$

y: the actual **target** value

ŷ : the output **predicted value**

i: the i^{th} **training sample**

acm acm.ai

Loss Function as a function of weights and bias

- The loss function depends on your predictions (**ŷ**)
- **ŷ** depend on the weights and bias of your model (**w_1, w_2 .... w_n , b**)
- The loss function can also be thought as a **function** of the **weights** and **bias** of your model!

$$\hat{y}(x) = b + w_1 x_1 + w_2 x_2 + \ldots\ldots w_n x_n$$

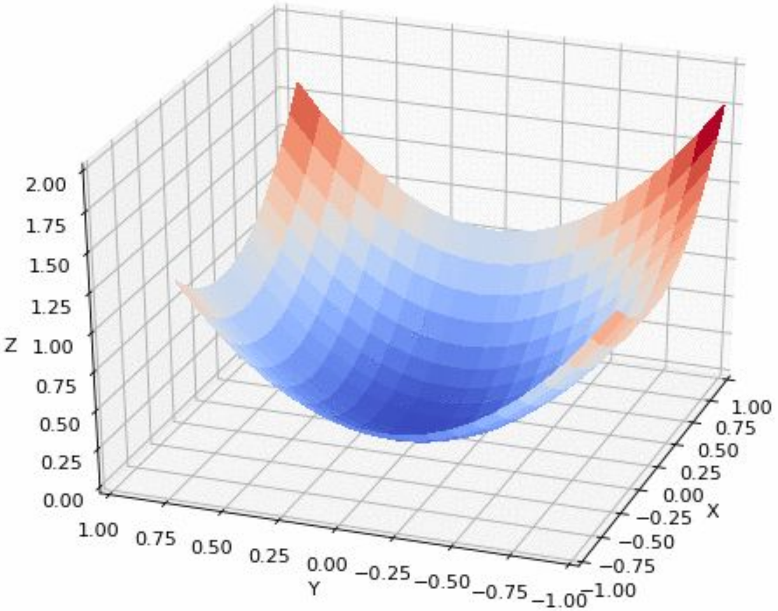
$$L(\hat{y_1}, \hat{y_2}, \ldots \hat{y_m}) = \frac{1}{m} \sum_{i=1}^{m} (y_i - \hat{y_i})^2$$

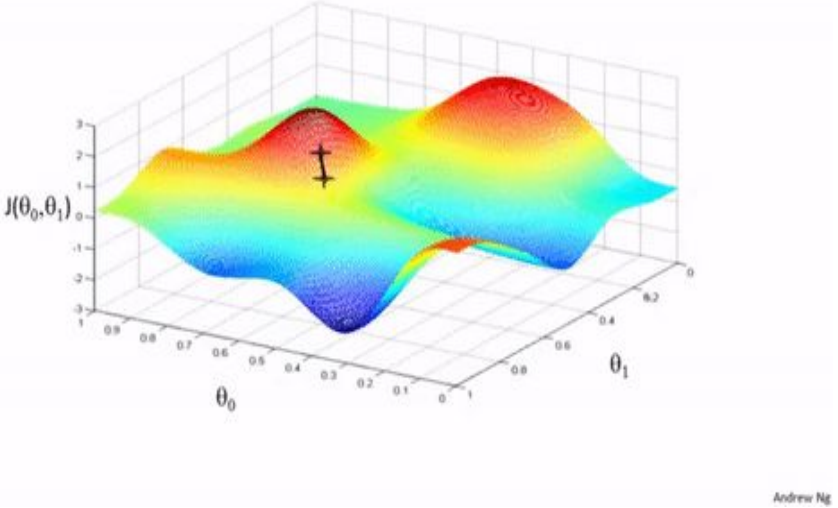
acm acm.ai

Learning Weights: Minimize Loss

- Our loss is a function of the **weights** and **bias** of our model
- Our model learns by updating its **weights** and **bias**
- If loss function outputs a small value → our model is **accurate**
- So, we need to find those **weights** for which the **loss is minimized**
- How do we do that?

Gradient Descent: How we minimize the value of a function

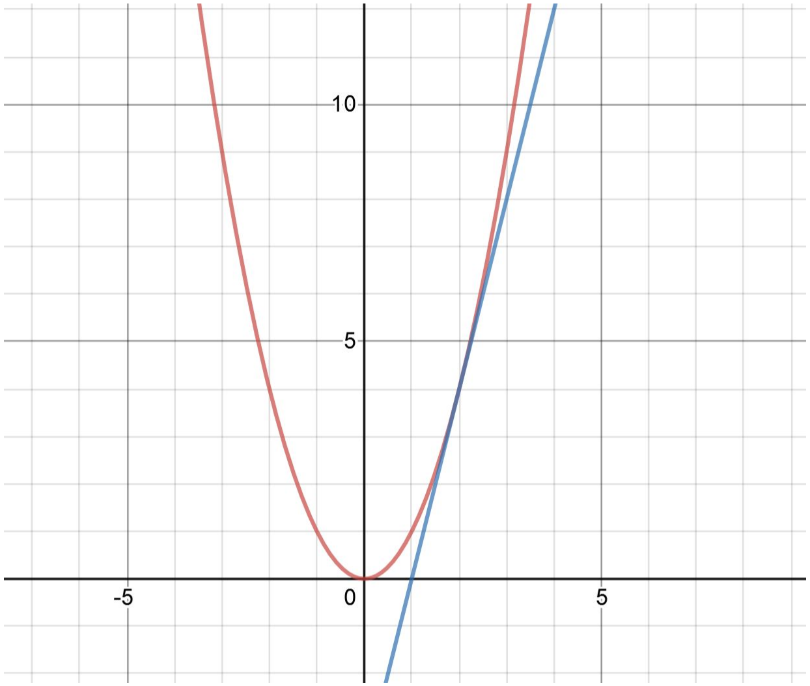




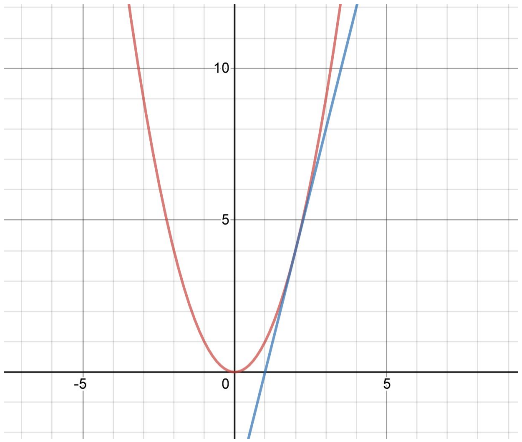
acm.ai

Single Variable Gradient Descent

- **f(x)** is a function of one variable: **x**
- **f'(x)**, the derivative, indicates whether the function is increasing or decreasing
- If **f'(x)** is positive, the function is increasing. So if **x** increases, **f(x)** increases
    - We want to decrease **f(x)** so we **decrease** x. i.e. we **subtract** *something* from x
- Similarly if **f'(x)** is negative, if we want to decrease f(x) we **increase** x



Single Variable Gradient Descent



To summarize: We want to **minimize** f(x), so

- if **f’(x)** is **positive**, we want to **subtract** *something* from x
- if **f’(x)** is **negative**, we want to **add** *something* to x

How do we do that?

- Use **f’(x)** itself!
- But careful! We want to do the “opposite” of what f’(x) tells us to

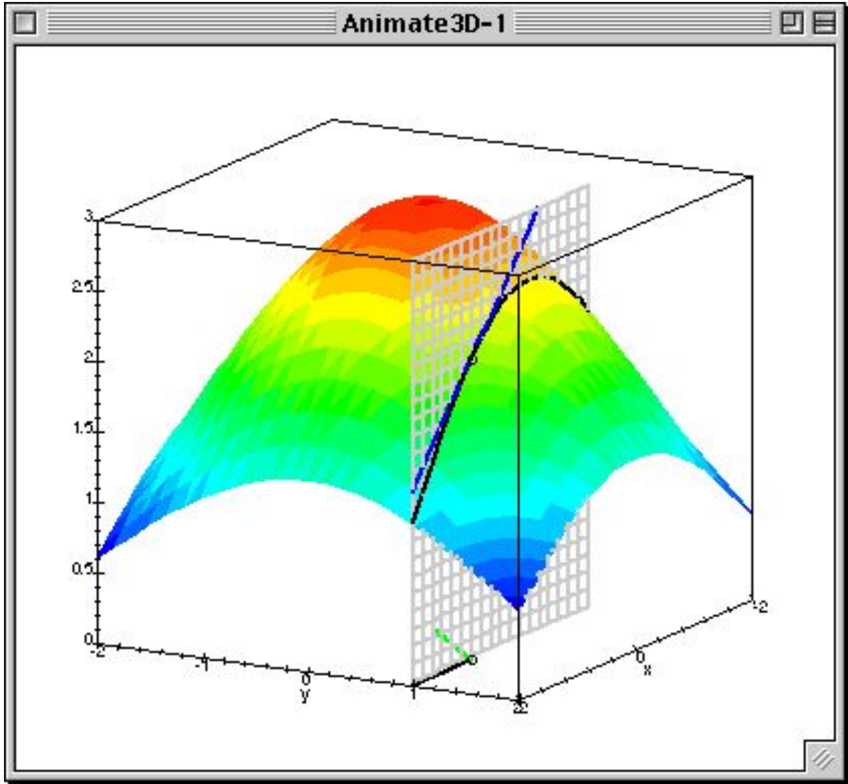
So we can **update** x like this $x = x - \alpha f'(x)$

alpha is just a constant we choose to scale f’(x). We call it the **learning rate.**

acm acm.ai

Multivariable Gradient Descent

- The gradient is the direction of **steepest ascent**
- Meaning that if we go in the same direction as the gradient we **increase** the value of the function
- But we want to **decrease** the value of the function
- So we go in the **opposite** direction as the gradient i.e. **gradient descent!**



Multivariable Gradient Descent

x is now a **vector**

$$\vec{x} = [x_1, x_2, \ldots . x_n]$$

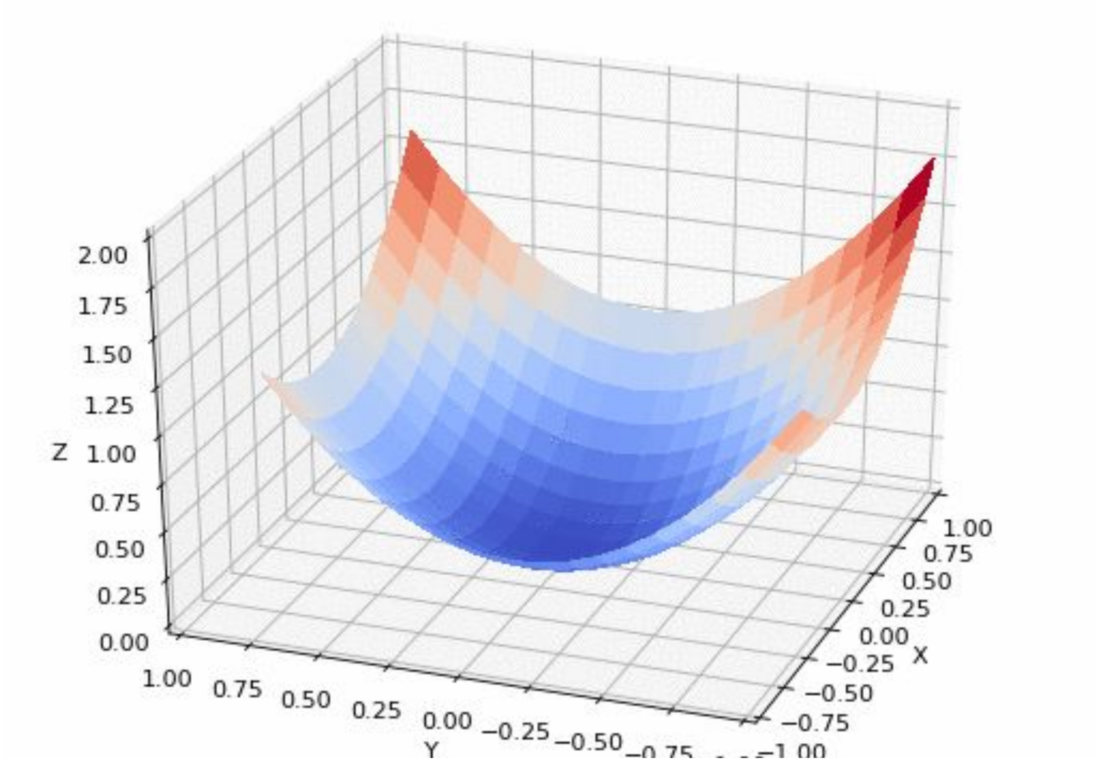
The **gradient** is also a **vector**

$$\nabla f(\vec{x}) = [\frac{\delta f}{\delta x_1}, \frac{\delta f}{\delta x_2} \ldots . . \frac{\delta f}{\delta x_n}]$$

So we **update** the x vector using the gradient vector

$$\vec{x} = \vec{x} - \alpha \nabla f(\vec{x})$$

Gradient Descent



Minimize loss using gradient descent

- Which function do we need to minimize? **The loss function**
- What are the weights of the function? $\mathbf{w_1, w_2 \ldots. w_n, b}$
- So the gradient we need to compute, is the gradient of the loss function with respect to $\mathbf{w_1, w_2 \ldots. w_n, b}$
- We can then update $\mathbf{w_1, w_2 \ldots. w_n, b}$ using these gradients

Minimize loss using gradient descent!

Taking the **gradient** of our MSE loss function

$$L(\hat{y_1}, \hat{y_2}, \dots \hat{y_m}) = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y_i})^2$$

$$\frac{\partial L}{\partial w_j} = \frac{2}{m}\sum_{i=1}^{m}(\hat{y_i} - y_i)x_{ij}$$

$$w_j = w_j - \alpha\frac{\partial L}{\partial w_j}$$

$$\frac{\partial L}{\partial b} = \frac{2}{m}\sum_{i=1}^{m}(\hat{y_i} - y_i)$$

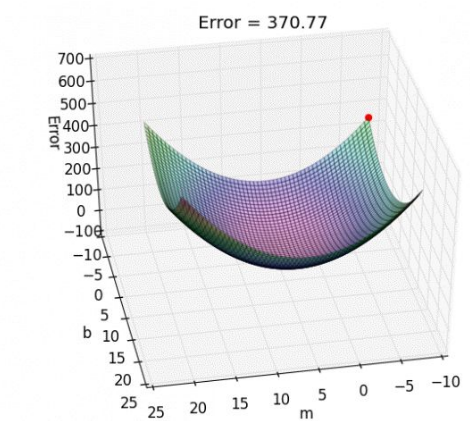
$$b = b - \alpha\frac{\partial L}{\partial b}$$

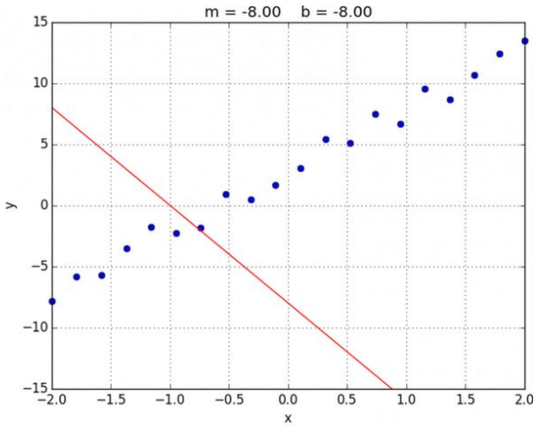
i refers to the **i**th training sample, **j** refers to the **j**th feature

Here's the full derivation of the gradients of MSE

Best Fit

- Minimizing the loss function can be thought of as finding the **best fit** “line” for your data





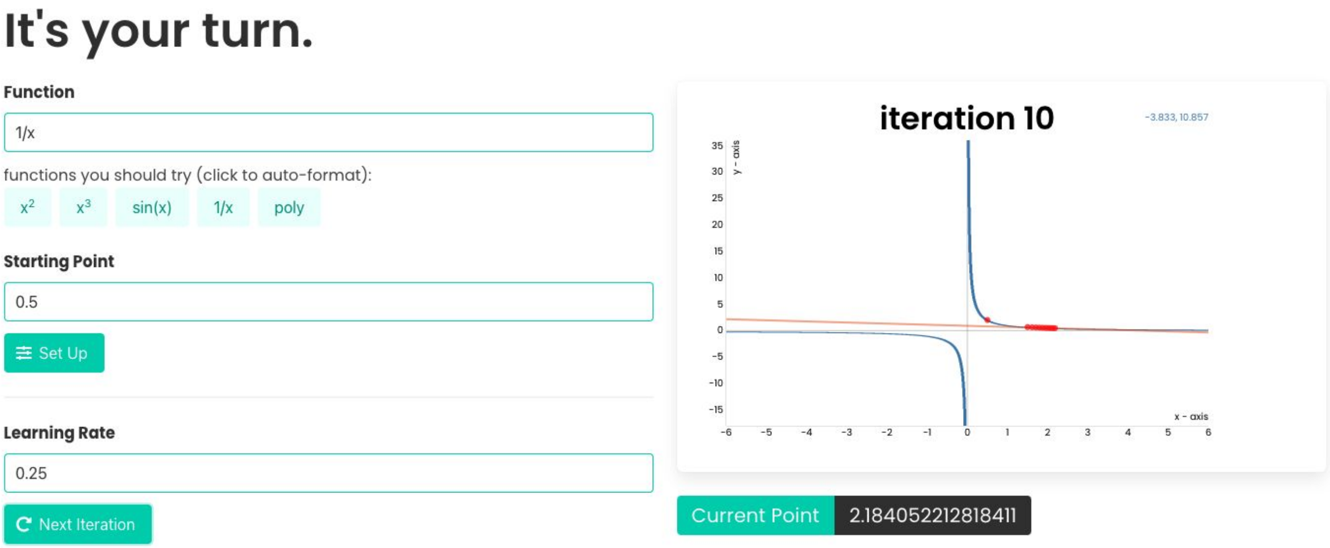
- This is linear regression with **one** feature. We are trying to fit a **line** with the given data.
- You will implement this from scratch in a project later in the quarter!

After we learn weights: Testing

- To test our model, we first select some of the data points we have that our model has **not** seen
- We then feed the **input features** of those data points into our model and keep aside the true **y** values
- Our model generates **predictions** using the input features
- We calculate the **loss** between our predictions and the true values
- And that loss tells us how well our model has performed!

Cool Single Variable GD Visualizer

https://uclaacm.github.io/gradient-descent-visualiser/



Implementing a Linear Regression Model with SKLearn

acm acm.ai

Linear Regression Coding Exercise

Follow Along at: tinyurl.com/btrack-w22-w3-demo

So there you have it

- Linear Regression is an extremely useful model and is the building block of pretty much **every single** machine learning, deep learning, and statistical model.
- The next topic to learn is Logistic Regression, where we'll be classifying objects, instead of predicting values.

Thank you! We’ll see you next week!

Please fill out our feedback form: bit.ly/btrack-w22-feedback

Next week: Logistic Regression

How does a computer recognise cats and dogs?

FB group: facebook.com/groups/uclaacmai